GPU Accelerated Intervisibility and Correlation with OneSAF Testbed

Dr. Guy Schiavone, Judd Tracy, Eric Woodruff, and Mathew Gerber

IST/UCF

University of Central Florida

3280 Progress Drive

Orlando, FL 32826

Troy Dere, Julio de la Cruz

RDECOM-STTC

Orlando FL 32826

Commoditization of Computing

- Mass market economics drives “Moore’s Law” : exponential increase in performance/cost ratio.
- Combining commodity hardware and free-source software can provide low-cost “supercomputing”: Beowulf clusters
- Graphical Processing Units (GPUs) progressing even faster (Super Moore’s Law)

Intervisibility Problem in CGF

- Dynamic Entity Interactions a major constraint on performance in CGF systems
- Hypothesis: Reducing time of Line-of-sight (LOS) calls can significantly increase number of supportable entities in CGF
- Idea – Combine cluster computing with GPU co-processing, test scalability.

Background

- 1994- Becker, Stirling: Beowulf Clusters
  - Highly successful for parallel processing problems with low communication overhead
- Late 1990's – GPU's used to solve alternative problems
  - 1998-2000 –Accelerated point visibility queries (Z-buffer queries)
  - UNC (Dr. Manocha) – Volume rendering, Collision detection… (Optimizing data structures, coordinating CPU/GPU processing)

Overview

- Compare performance using “generic” CTDB and OpenFlight Formats
  - High-Level API – OpenSceneGraph (OSG)
  - Free source – Extensible, Rapid Prototyping
  - Active Community – Well Supported, Efficient Implementation
  - Forces the use of an Update/Cull/Render paradigm
- Compare LOS Correlation between SAF and Visual Simulation

Visualization Algorithm

- Uses OpenGL extension called NV_Occlusion_Query (NVidia, ATI, MESA 6.0)
  - allows query of the graphics hardware of how many pixels are rendered between the time a begin/end pair occlusion query call are performed
  - originally created to determine if an object should be rendered
  - our algorithm takes advantage of it to see what percentage of an entity is actually rendered

Update stage

- Update stage of the scene graph is where all data modifications are made that affect the location and properties of objects in the scene graph
  - entities positions and orientations are updated along with all sensor orientations
  - scene graph is traversed and the distance between each sensor and all entities is calculated
  - algorithm has one call to the Update stage per time step

Cull Stage

- all geometry is checked against a view frustum to determine if is should be rendered.
  - We apply Area-of-Interest (AOI) to further cull entities
- For this algorithm the render order is critical: All terrain and static objects should be rendered first as they always potential occluders.
- Next all entities and dynamic objects are rendered in a front to back order (visibility of entities not occluded by closer objects)

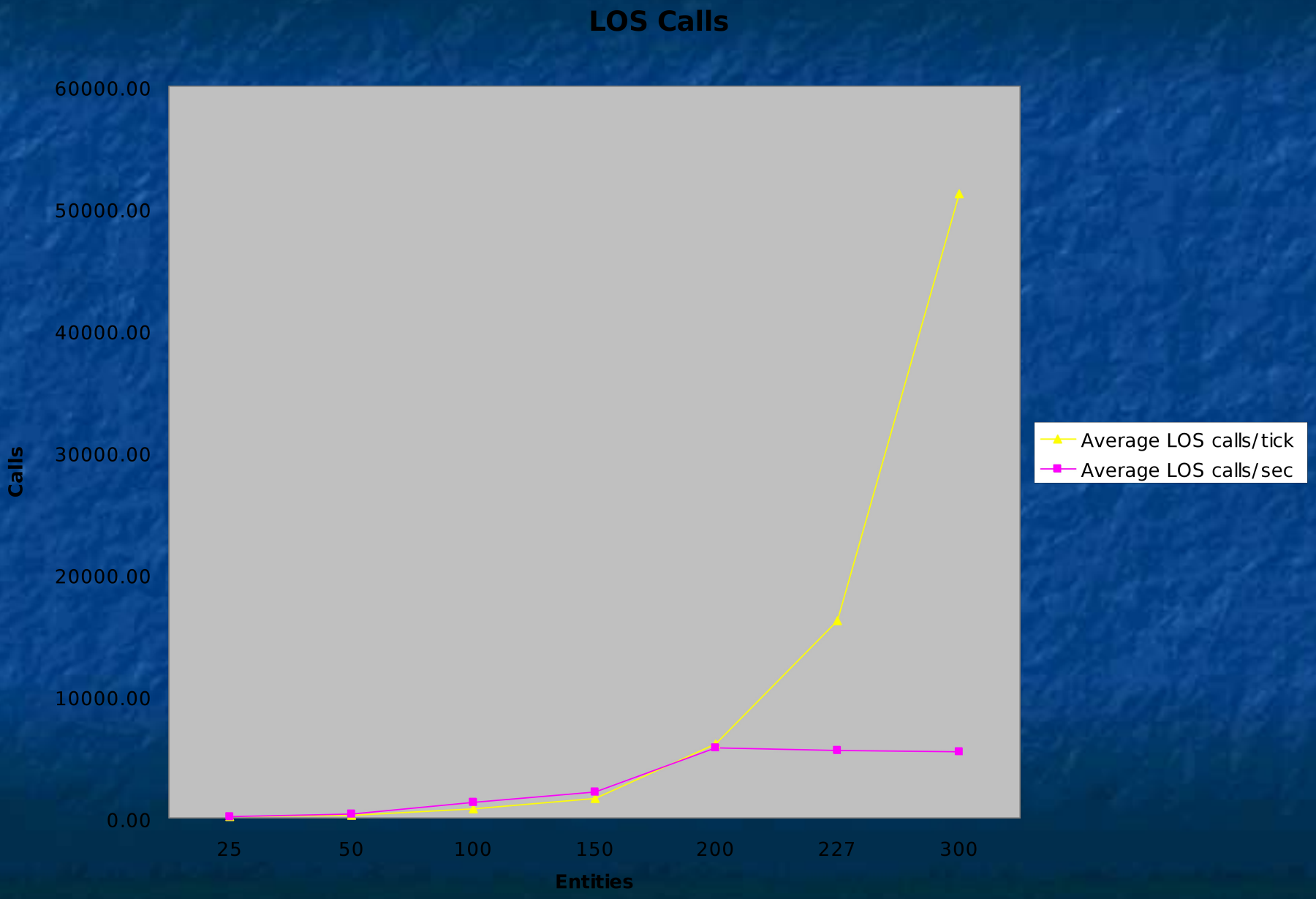
Render stage

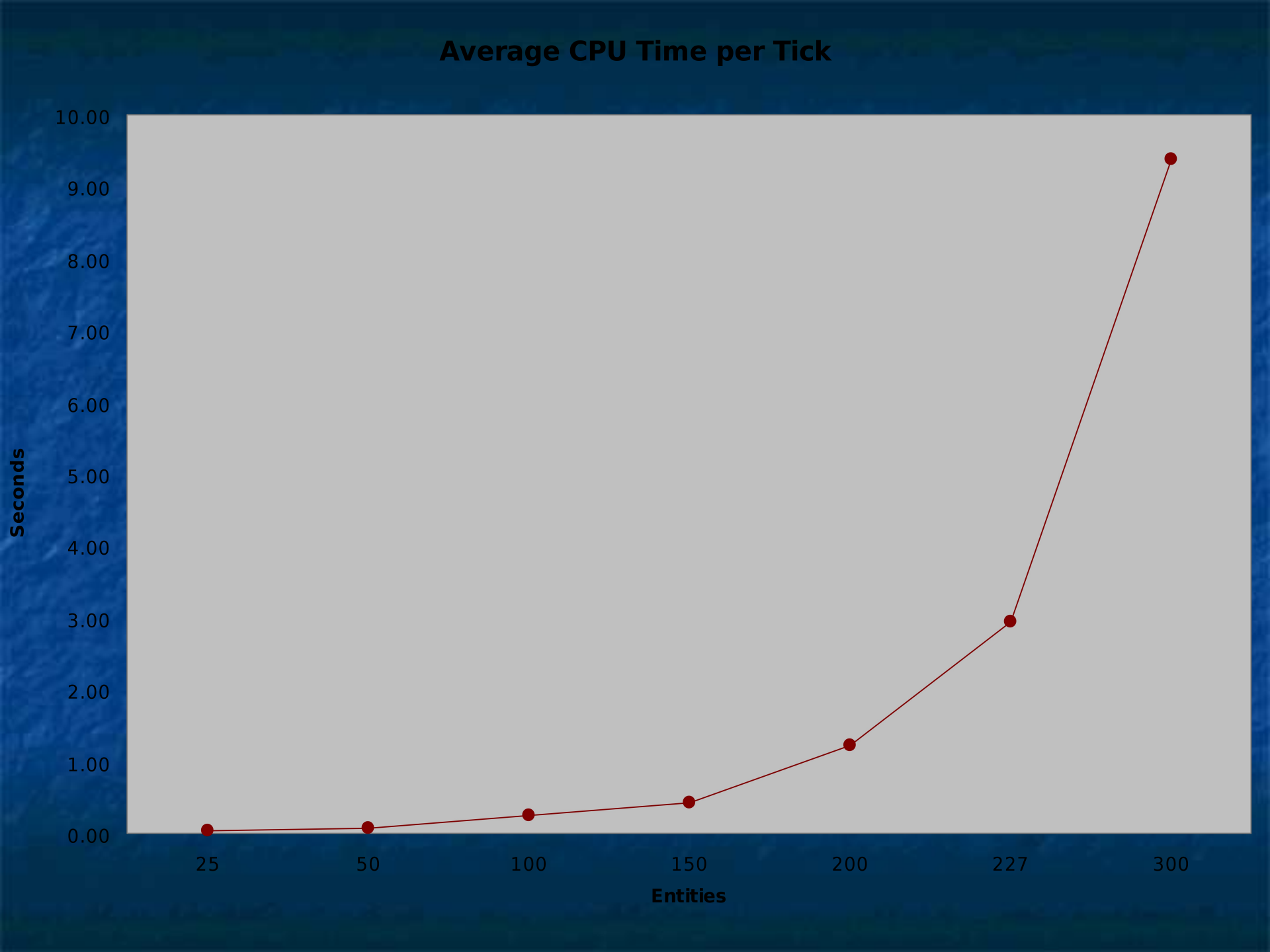
- All terrain and static objects are rendered first
- Each entity is rendered twice in front to back order wrapped with NV_Occlusion_Query begin/end calls
  - first time an entity is rendered the depth buffer and color buffers are disabled to obtain the amount of pixels an entity uses with out being occluded
  - entity is rendered again with the depth and color buffers enabled to obtain the amount of pixels actually visible
  - Intervisibility = visible pixels/total pixels per entity

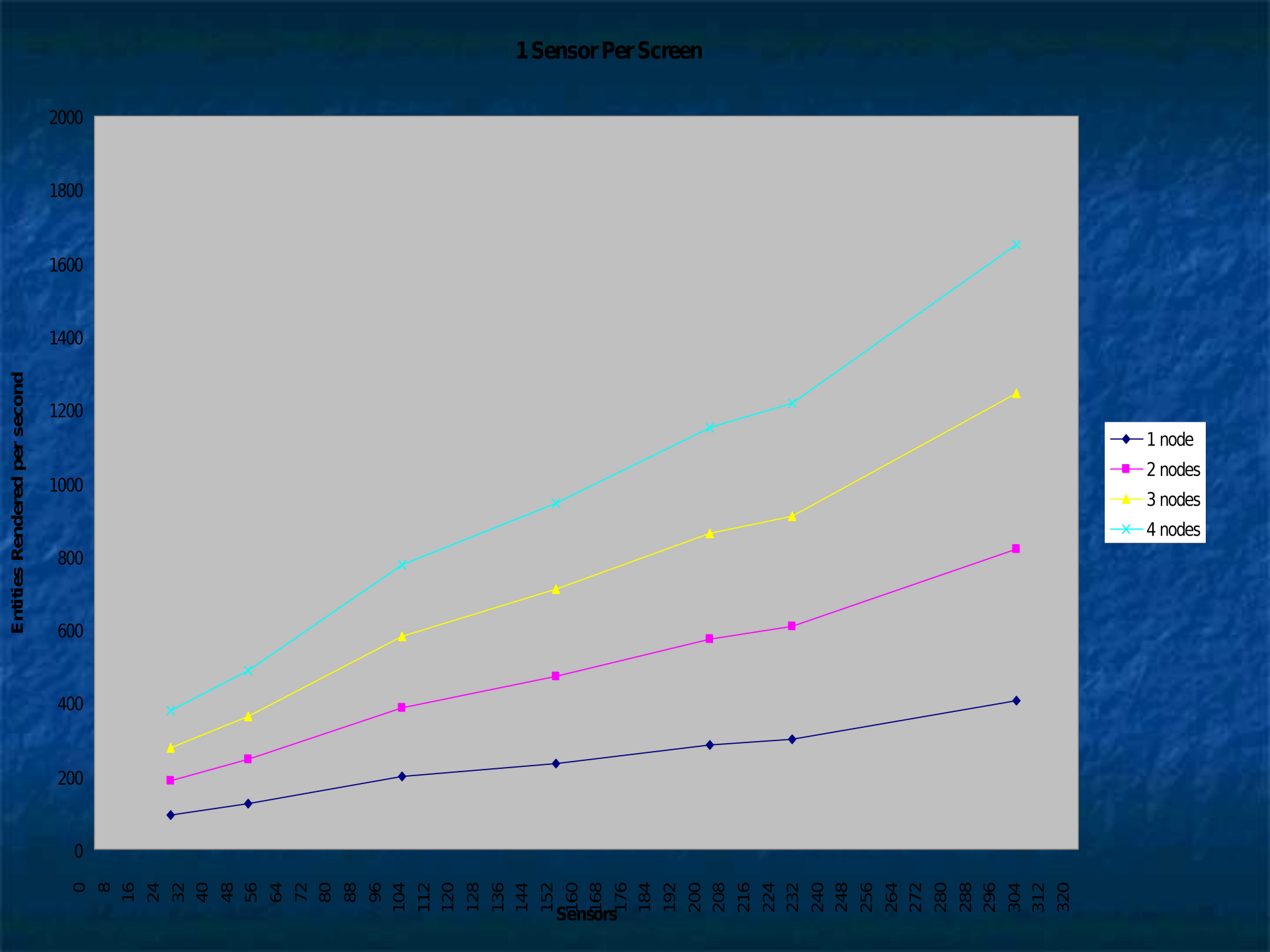
Hardware Specs

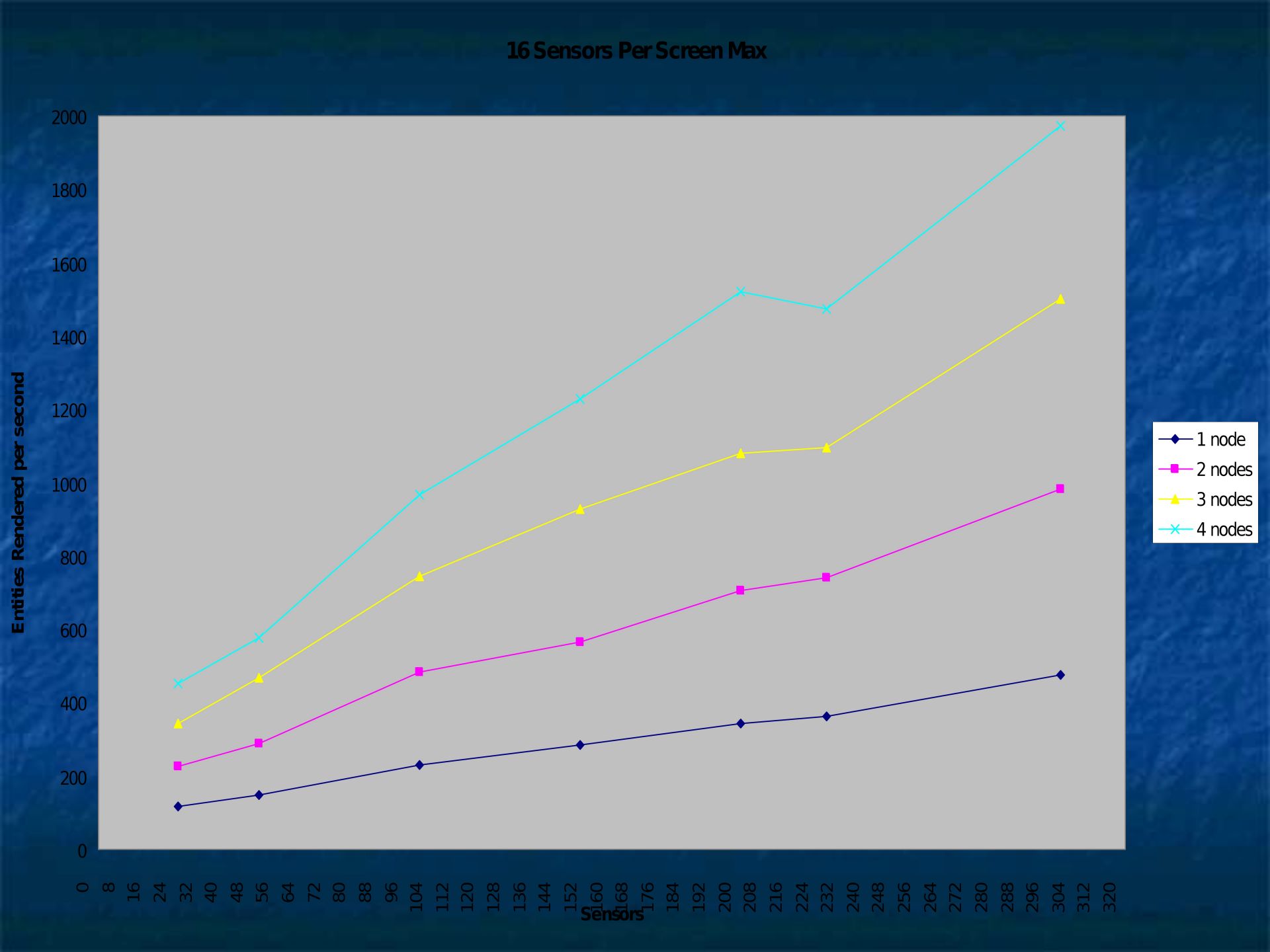
- Compute Node – Dual AMD Athlon 1.33 GHZ, 512 MB RAM, Fast Ethernet network
- GPU - NVIDIA GeForce FX5900 Chipset
  - 256MB DDR SDRAM
  - 400 MHz engine clock
  - 850 MHz memory clock
  - 400 MHz internal RAMDAC
  - 300 Million vertices/ sec
  - 3.6 Billion texels/ sec fill rate
  - 27.2 GB/sec memory bandwidth
  - 8 pixels per clock rendering engine

SAF Performance

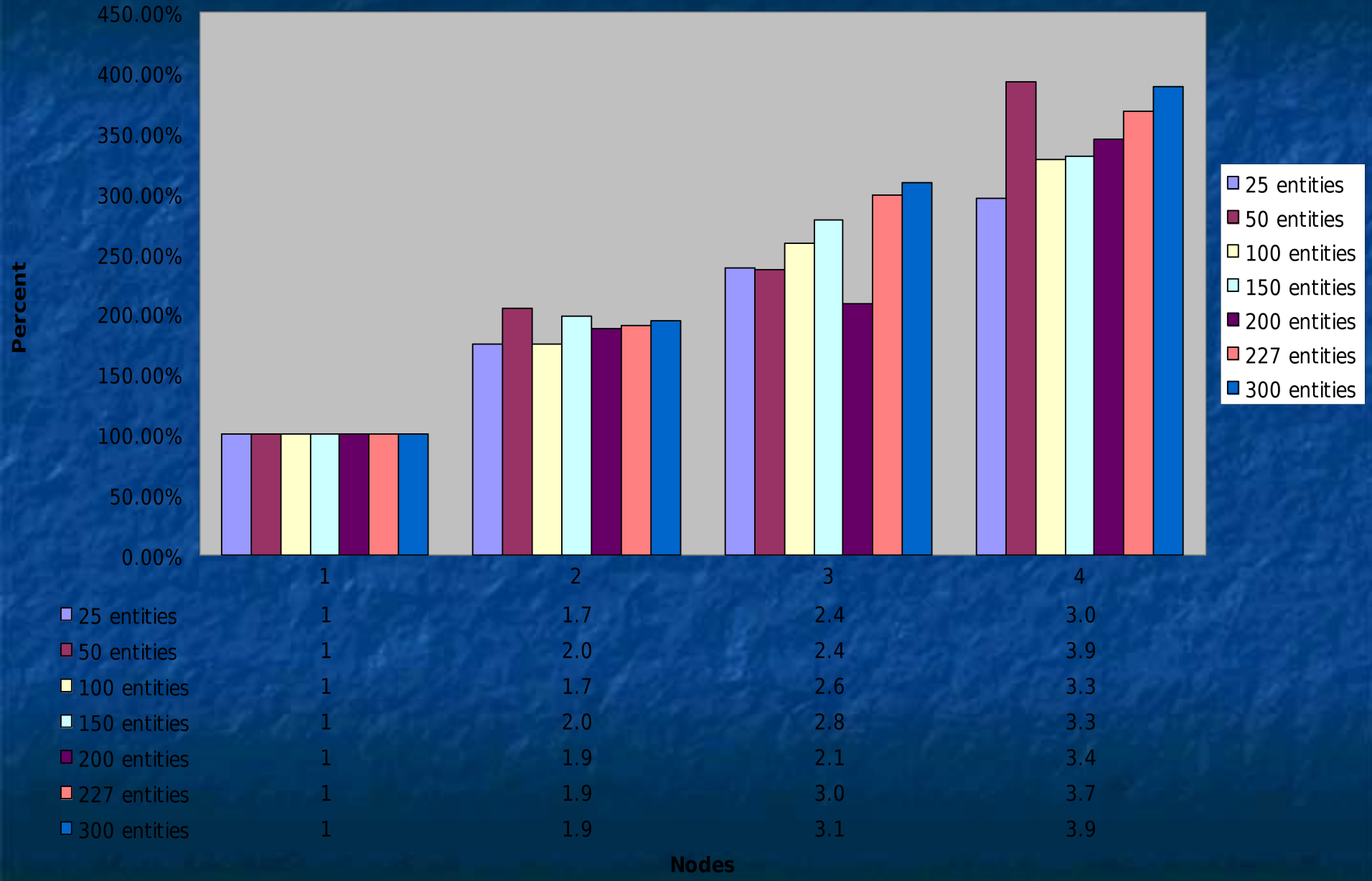



Average CPU Time per Tick
10.00
9.00
8.00
7.00
6.00
5.00
4.00
3.00
2.00
1.00
0.00
Seconds
25
50
100
150
200
227
300
Entities

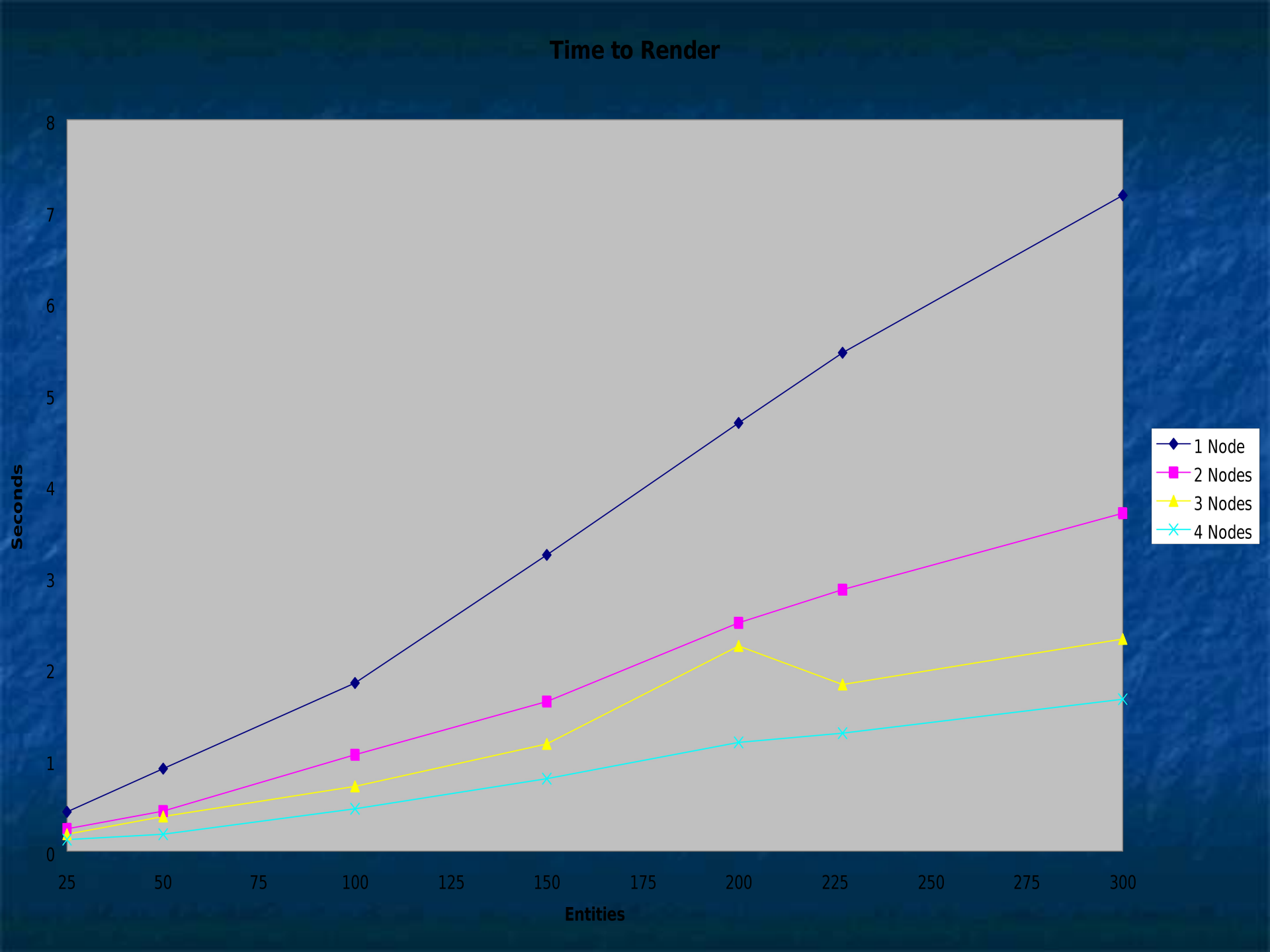

1 Sensor Per Screen
Entities Rendered per second
Sensors
1 node
2 nodes
3 nodes
4 nodes


16 Sensors Per Screen Max
Entities Rendered per second
Sensors
0
200
400
600
800
1000
1200
1400
1600
1800
2000
0
8
16
24
32
40
48
56
64
72
80
88
96
104
112
120
128
136
144
152
160
168
176
184
192
200
208
216
224
232
240
248
256
264
272
280
288
296
304
312
320
1 node
2 nodes
3 nodes
4 nodes

Percent of Single-Node Performance



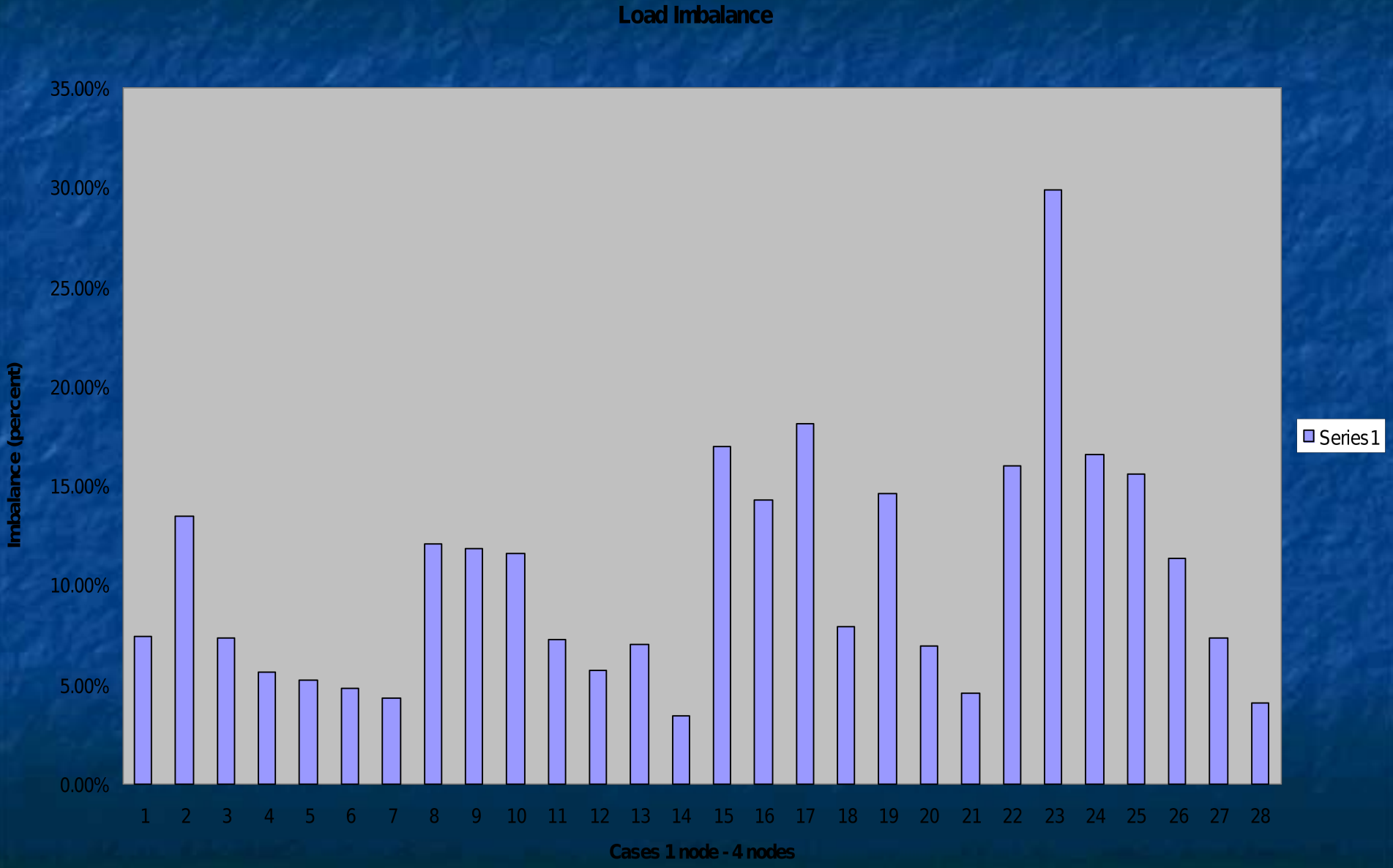
	1	2	3	4
25 entities	1	1.7	2.4	3.0
50 entities	1	2.0	2.4	3.9
100 entities	1	1.7	2.6	3.3
150 entities	1	2.0	2.8	3.3
200 entities	1	1.9	2.1	3.4
227 entities	1	1.9	3.0	3.7
300 entities	1	1.9	3.1	3.9

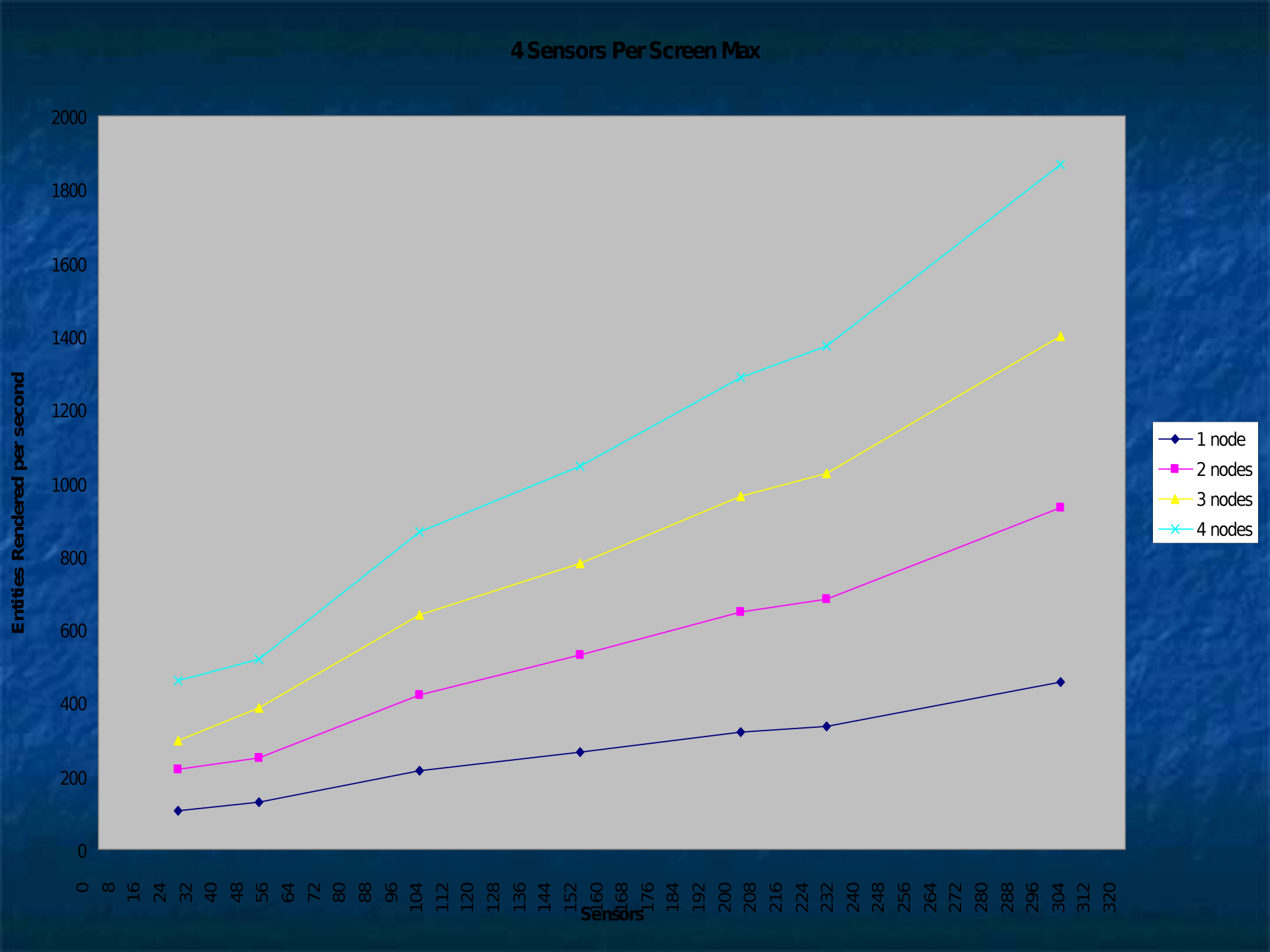

Time to Render
Seconds
0
1
2
3
4
5
6
7
8
25
50
75
100
125
150
175
200
225
250
275
300
Entities
1 Node
2 Nodes
3 Nodes
4 Nodes

Distributed Calculations

- Front end distributes entitles at start in random order
- Preliminary algorithm - No load-balancing
- Load Imbalance ranges from 4% -30 %
- Current approach “embarrassingly parallel”
- Each Node has full database
- Load Balancing optimization must have minimal communication overhead (global)

Load imbalance Example – 1 sensor/screen, 1-4 Nodes, 7 Cases




4 Sensors Per Screen Max
Entities Rendered per second
Sensors
0
200
400
600
800
1000
1200
1400
1600
1800
2000
0
8
16
24
32
40
48
56
64
72
80
88
96
104
112
120
128
136
144
152
160
168
176
184
192
200
208
216
224
232
240
248
256
264
272
280
288
296
304
312
320
1 node
2 nodes
3 nodes
4 nodes

Terrain Database Correlation

- Initial Problems:
  - Lack of coordination between differing simulation applications in the approach to terrain generation process
  - Differences in source data
  - Differences in resolution required for various interoperating applications
  - Proprietary competition for dominance in formats
  - Lack of common agreement in a necessary and sufficient set of features and attributes
  - A lack of tools for assessing synthetic environment correlation

Progress

- SEDRIS
  - EDCS
  - Spatial data Reference Model
  - STF
- Assessment Tools:
  - See-It
  - Side-by-Side Viewer
- Standard Commercial and “freeware” convertors

Problems Still Exist

- Inherent differences in Visual Simulation and CGF data structures, formats and processing
  - Example: topology explicit in CGF, but “implied” by contiguous texture in visual systems.
  - Visual system geometry inherently more complex and explicit, CGF can handle feature geometry within algorithms

Our Focus – Intervisibility Correlation

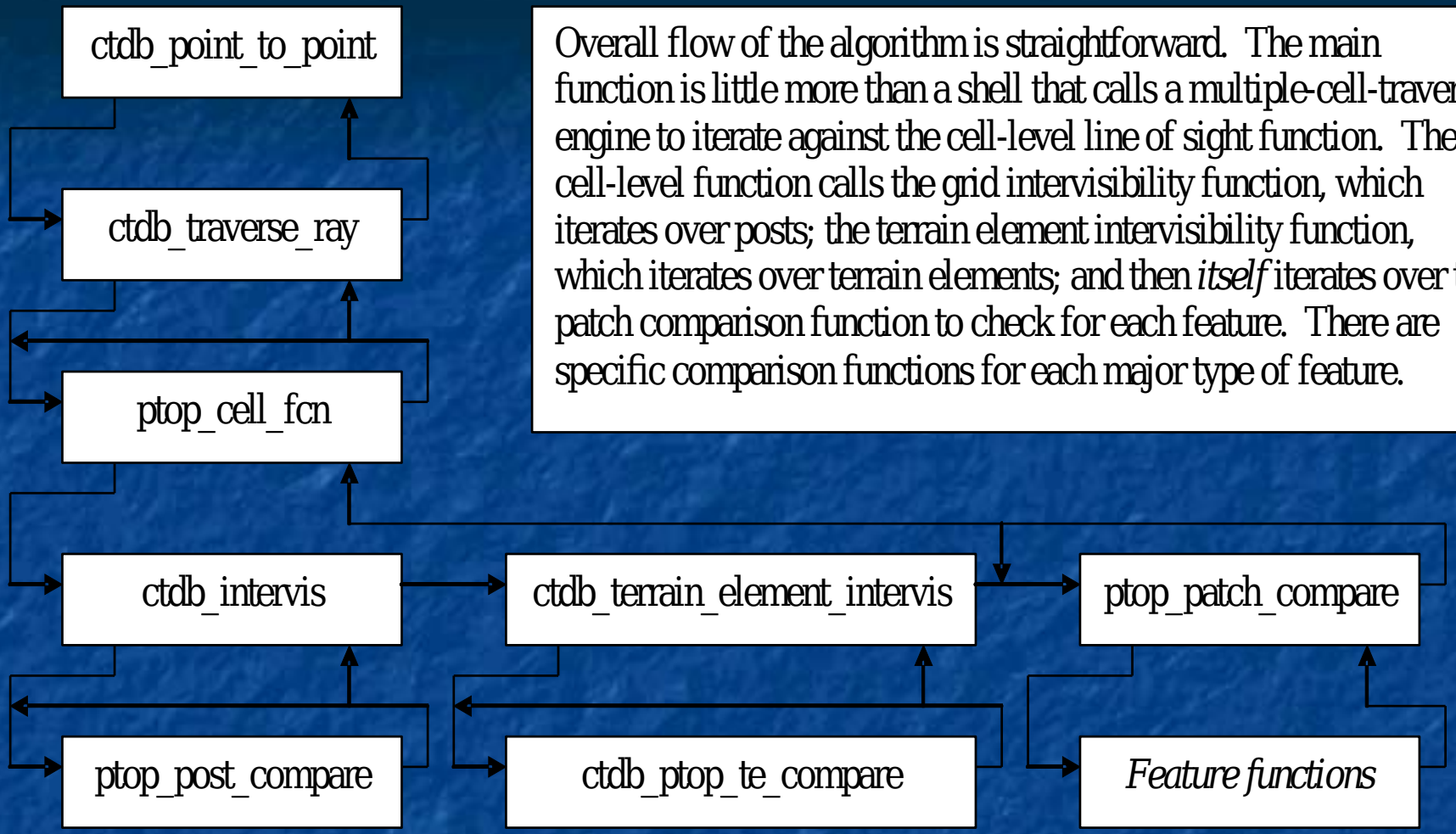
- Objective – provide techniques to assess intervisibility correlation between applications employing CTDB and OpenFlight data
- “Example” applications used: OTB and OSG
- Initial goal – Assess need for such a tool: measure intervisibility correlation between CTDB and OpenFlight Databases

Approach

- Employ GPU-Accelerated techniques described in previous paper to determine visual system intervisibility
- Test results against OTB in geo-specific database to determine “trouble spots”
- Integrate needed libctdb components into an application to compare correlation

OTB Intervisibility Calculations

- More Complex, “case-by-case”
- Key function call to determine intervisibility is ctdb_point_to_point() [libsrc/libctdb/ct_ptop.c].
- Not properly point-to-point, but point-to-cone
- core parameters:
  - A starting point,
  - An ending point, and
  - A width for the cone at the ending point.
- Optional parameters:
  - A minimum visibility level at which to “cut off”,
  - Functions to specify the opacity of linear features, and
  - A list of vehicles to test for visibility blocking



Overall flow of the algorithm is straightforward. The main function is little more than a shell that calls a multiple-cell-traversal engine to iterate against the cell-level line of sight function. The cell-level function calls the grid intervisibility function, which iterates over posts; the terrain element intervisibility function, which iterates over terrain elements; and then *itself* iterates over the patch comparison function to check for each feature. There are specific comparison functions for each major type of feature.

CTDB LOS and Cell Composition

Key functions ctdb_point_to_point, ctdb_traverse_ray

Iterates on the **Cell Check** to find intervisibility values across terrain cells traversed by the LOS ray, and composes the separate results into a single set of numbers. If the terrain feature check ever uses raster-based computation, the final raster is used to modify the visible area returned here. Can also check for entities blocking other entities from view; we are not considering this in our current analysis. Returns **visible area** and **linear transmittance**.

Cell Check

Key function ptop_cell_fcn

Checks the blockage of the intervisibility ray within a single cell. If grid terrain is present then the **Grid Terrain Collision Check** is called first; otherwise, the **TINned Terrain Collision Check** is called first and the grid check is still performed to catch features. **Terrain Feature Check** is called last.

Grid Terrain Collision Check

Key function ctdb_intervis

Performs two functions:

- Checks for collision of LOS ray with gridded terrain patches.
- Checks for interference with LOS ray by terrain features (trees, buildings, etc.) along gridded terrain patches and adds these features to the terrain feature list.

TINned Terrain Collision Check

Key ctdb_terrain_element_intervis

Checks for collision of LOS ray with TINned terrain elements.

Terrain Feature Check

Iteration of ptop_patch_compare, multiple feature-specific functions

Checks for effect on LOS ray of terrain features from the terrain feature list.

- Buildings are handled by rasterization.
- Trees can be handled either by raster-based drop in visible area or by statistical drop in linear transmittance (OTB always, so far as we can tell, does the latter).

Some Likely Areas of Difference (1)

- Cases of hilly terrain blocking the target – cases in which terrain causes blockage from the side, or in which the height of the terrain as it interferes with the LOS ray is not the average height of the terrain as it interferes with the the LOS rectangle, will reveal significant discrepancies between our visual LOS readings and libctdb’s

Some Likely Areas of Difference (2)

- Any case that relies on the shape of the target may display significant discrepancies between our LOS readings and libctdb's, either due to libctdb using a simple rectangle or due to the small size of libctdb's rasters

Some Likely Areas of Difference (3)

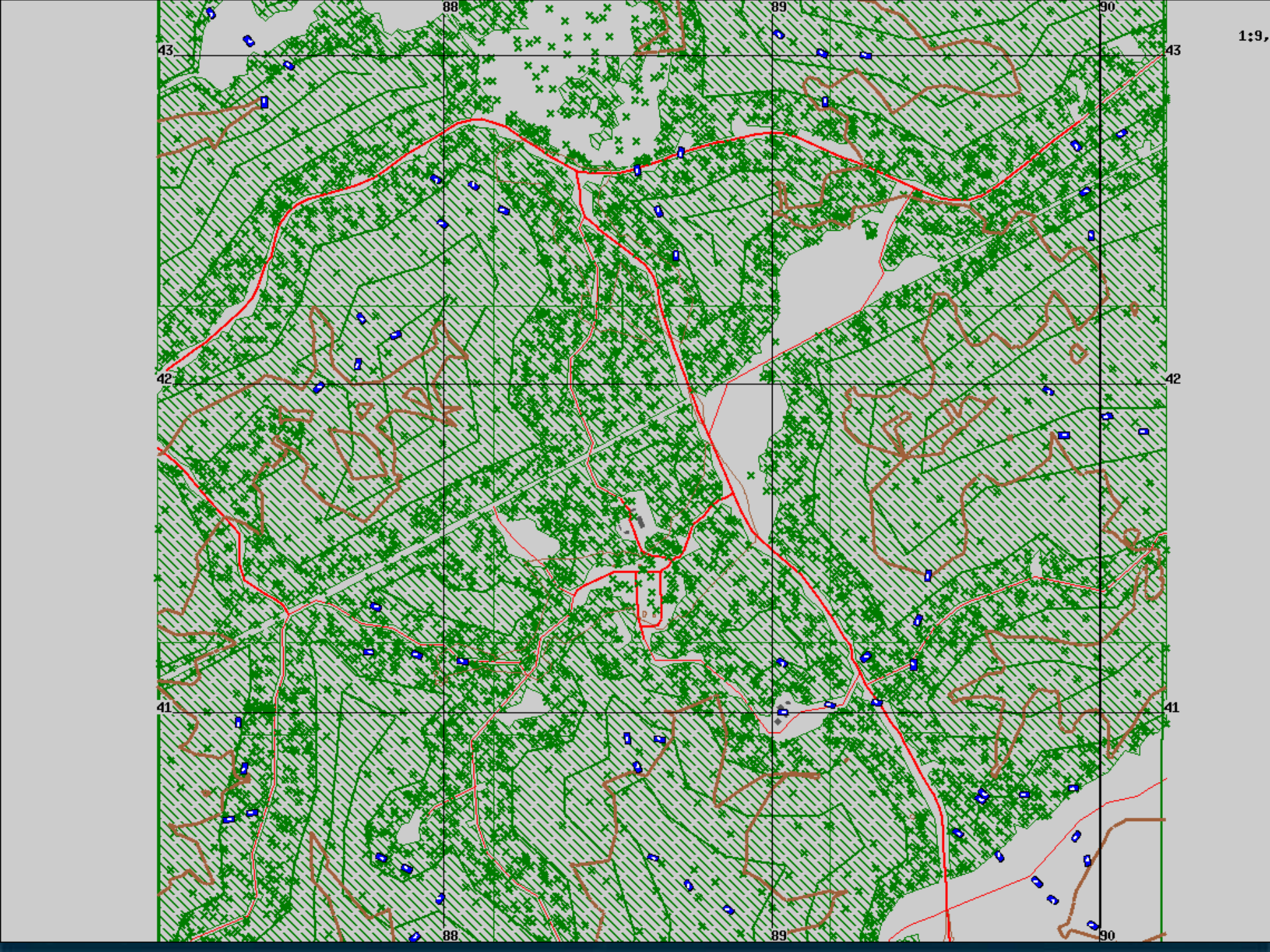
- Any case that involves line of sight through trees may display significant discrepancies between our LOS readings and libctdb's. In non-raster mode, if two tree trunks block a target, it is easily possible for libctdb to report about 25% visibility when the target in fact cannot be seen at all! In raster mode, the roughness of the tree models used will come into play. For foliage, transmittance = (1 - w_area) *(1 - h_area) * fullness

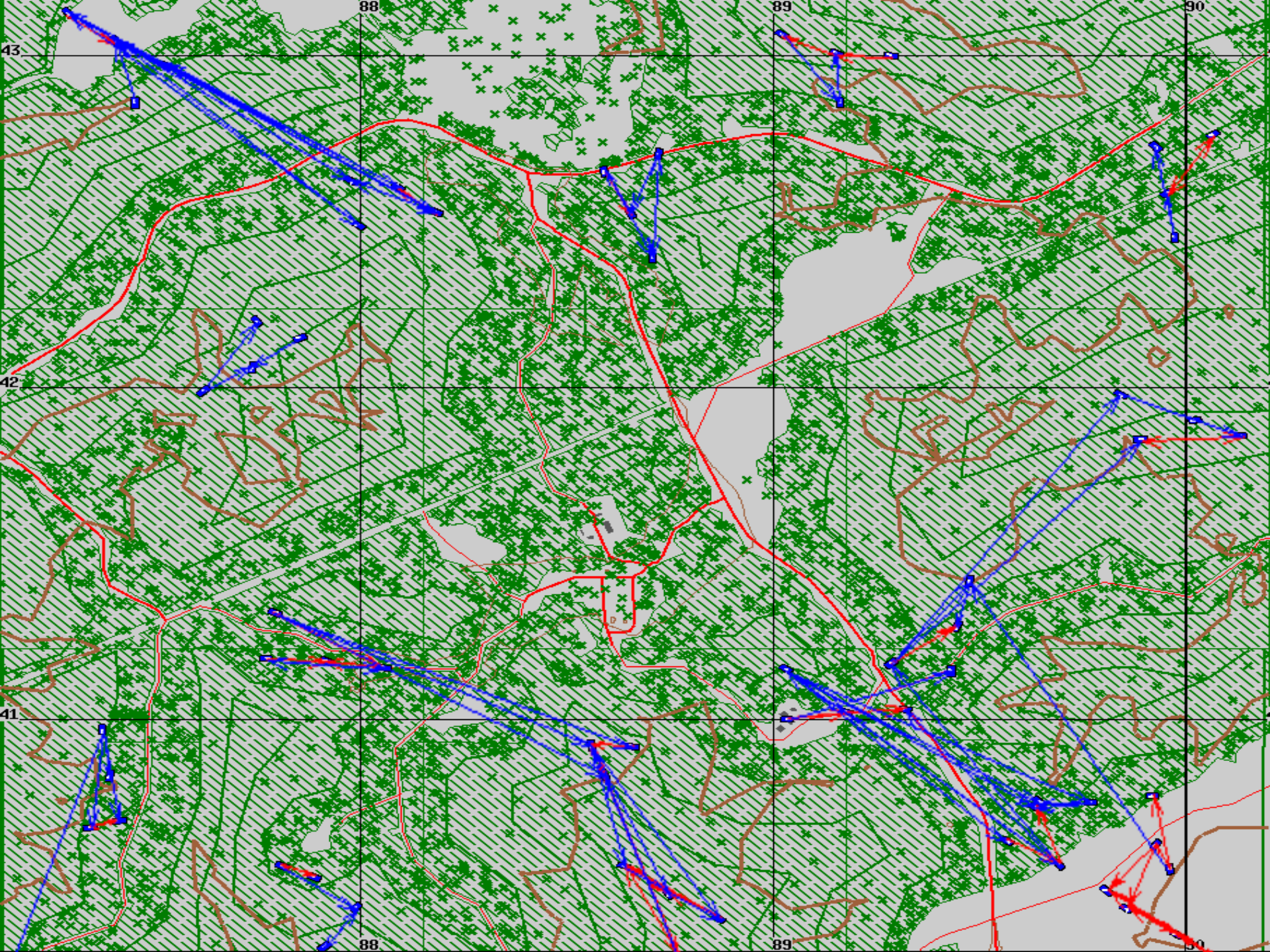
Test Case

- Database ftpolk_r10-0000 created Wed Jul 17 10:37:15 2002

  Terrain Format 7 with the following features:
  UTM flat, MIXED TIN & GRID POST,

  Grid Spacing (METERS): 48
  Fixed point basis : 0.01907349
  Origin at 3440189N 487129E in UTM zone 15R (datum WGS84)
  Minimum (SW Corner)(X,Y) : (0.00, 0.00)
  Maximum (NE Corner)(X,Y) : (3072.00, 3072.00)
  Minimum Elevation : (83.52)
  Maximum Elevation : (132.35)


1:9,
88
89
90
43
42
41


88
89
90
43
42
41
88
89

Correlation Analysis

	CTDB_VIS	*GPU_VIS*
CTDB_VIS	1	
GPU_VIS	0.5184368	1

Kappa	
P1	0.70319874
P2	0.00036875
K	0.70308926

Mean Squared Difference	0.04939703

Conclusions – GPU Acceleration

- Use of multiple GPUs a scalable approach, with potential performance on the order of OTB
- Parallelization/GPU effective, parallezation/screen requires geometry LOD adjustment
- Approach has potential employment as “Intervisibility Server”.

Conclusions - Correlation

- The major problem found is correlation through tree lines and foliage
- OTB treats intervisibility as a transmittance, while visual systems can use geotypical or geospecific actual tree textures
- For geotypical case, only statistical correlation can be obtained
- For geospecific case, an intervisibility server operating on a common data source is recommended

Future work

- Implement Load balancing
- Optimize multiple sensor/screen cases by Level-of-detail adjustments
- Extend GPU cluster results to 16
- Design and Implement Data Structure Optimizations
- Texture and Geometric Analysis for improved fidelity
- “Perceptual” Image Processing to determine target detection and recognition